LED blinker

Introduction to digital Low-Level Radio Frequency Controls in Accelerators

Lab 3

Qiang Du

US Particle Accelerator School
Janurary 23 – 27, 2023

Contents

1 Introduction

In this lab we will learn the basics of FPGA programming, by blinking the 4 LEDs on either the Digilent Arty A7 board, and the LBNL LLRF controller.

Here we provide working examples from basic RTL simulation to synthesize, layout, bitstream file generation,

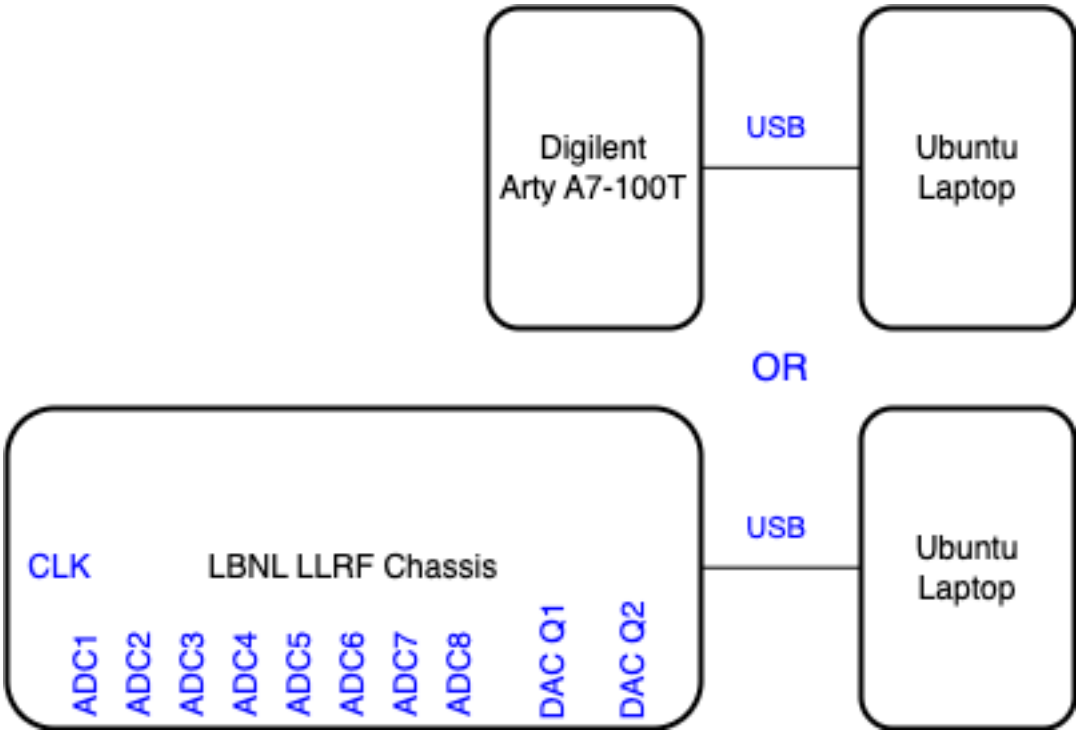
The goal of this lab is for you to get familiar with:

1. RTL design and verification
    1. Get familiar with GNU make using `Makefile`s;
    2. Write a simple LED blinker in `verilog`;
    3. Write behavioral test bench;
    4. Build test bench and run simulation;
    5. Examine waveforms and understand logic behavior;
2. Synthesize the design
    1. Understand the synthesizing process;
    2. Write a `vivado` tcl script to define the process;
    3. Write position constraint file;
    4. Write timing constraint file;
    5. Execute synthesize, observe timing and utilization report;
3. Test on hardware
    1. Program the bitstream file on an FPGA device;
    2. Observe the LED blinker behavior and compare with simulation;

1.1 Glossary

- **FPGA** Field-Programmable Gate Array.
- **RTL** Register Transfer Level design abstraction for a digital circuit design such as an FPGA.
- **GNU** Stands for "GNU's Not Unix", a collection of open source software tools that led to the family of operating system such as Linux.

1.2 Hardware setup



2 Exercise

2.1 RTL logic

The logic of this LED blinker is simply a 32 bit counter, in `led_test.v`:

```
module led_test #(
    parameter MSB = 27
) (
    input clk,
    input reset,
    output [3:0] led
);
reg [31:0] cnt=0;
always @(posedge clk) begin
    cnt <= reset ? 32'h0 : cnt + 1'b1;
end
assign led = cnt[MSB:MSB-3];
```

If the frequency of `clk` is 100 MHz, what's the expected LED blinking rate for each bit?

2.2 Build test bench

We use an open source simulator Icuras Verilog which is available on most Linux / MacOS platforms.

To build the testbench `led_test_tb` from source file `led_test_tb.v` and `led_test.v`, run

```
iverilog -Wall -Wno-timescale -o led_test_tb led_test_tb.v led_test.v
```

Similar to most compliers, this building process is defined by a rule in `labs/rules.mk`, and now we can just run

```
make led_test_tb
```

2.3 Run behavioral simulation and build waveform

Now the executable `led_test_tb` is ready to run, and we do it by `vvp`, with argument of `+vcd` so that it saves the history of logics in the form of *waveforms* that can be examined later. The command is

```
vvp -N led_test_tb +vcd
```

or by `make`:

```
make led_test.vcd
```

2.4 Examine waveform

We use another open source tool `gtkwave` to view the waveform file, by

```
gtkwave led_test.vcd led_test.gtkw
```

where `led_test.gtkw` is the settings of the view. Or we can do it by `make`,

```
make led_test_view
```

Note this won't work on a MacOS, where one need to manually launch `gtkwave` app, unless an additional command line tool is installed.

2.5 Synthesize:

Two `tcl` scripts (`synth_*.tcl`) are provided to send things to Xilinx `vivado` for synthesizing, including `led_test.v`, the constraint file that describes pin mapping and timing and a top level verilog file. Try:

- for Arty A7

```
make arty_led_top.bit
```

- for Marble

```
make marble_led_top.bit
```

As a result of the last step in the `tcl` script, the bitstream file is generated, which we will test on hardware next.

There are co-product of the synthesizing process: the utilization and timing reports are generated in the log on the terminal. For example, for arty A7,

```
2. Slice Logic Distribution
---------------------------
+-------------------------------------------+------+-------+-----------+-------+
|                 Site Type                 | Used | Fixed | Available | Util% |
+-------------------------------------------+------+-------+-----------+-------+
| Slice                                     |    7 |     0 |     15850 |  0.04 |
|   SLICEL                                  |    7 |     0 |           |       |
|   SLICEM                                  |    0 |     0 |           |       |
| LUT as Logic                              |    1 |     0 |     63400 | <0.01 |
|   using O5 output only                    |    0 |       |           |       |
|   using O6 output only                    |    1 |       |           |       |
|   using O5 and O6                         |    0 |       |           |       |
| LUT as Memory                             |    0 |     0 |     19000 |  0.00 |
|   LUT as Distributed RAM                  |    0 |     0 |           |       |
|   LUT as Shift Register                   |    0 |     0 |           |       |
| Slice Registers                           |   28 |     0 |    126800 |  0.02 |
|   Register driven from within the Slice   |   28 |       |           |       |
|   Register driven from outside the Slice  |    0 |       |           |       |
| Unique Control Sets                       |    1 |       |     15850 | <0.01 |
+-------------------------------------------+------+-------+-----------+-------+

Timing Report

Slack (MET) :             7.214ns  (required time - arrival time)
  Source:                 led_test_i/cnt_reg[1]/C
```

There is also a file `arty_led_vivado.v` generated by vivado, where our `led_test` module is instantiated into pure Xilinx primitives such as D flip-flop and look up tables, a real digital circuit that is translated from our HDL code. We can mix primitive and logic HDL in our design, for example there are input buffer `IBUF` and clock buffer `BUFG` in `arty_led_top`.

Our LLRF HDL library tries to be open source and platform independent as much as possible.

2.6 Program bitstream file

Connect your board and the command for programming bit file is:

- for Arty A7 `bash  make config_arty_led`
- for the LLRF chassis `bash  make config_marble_led`

Observe the LED blinking frequency and compare with your expectations. The clock frequency is specified in the constraint file of each board.

Show the blinking LED on either board.

2.7 Your turn: make the LED dimmable

Now try to update the `led_test.v` and make the 4 dimmable LEDs using Pulse-Width Modulation. Explain brightness.